

# Parameterized Synthesis Case Study: AMBA AHB (extended version)\*

Roderick Bloem

Swen Jacobs

Ayrat Khalimov

Graz University of Technology, Austria <sup>†</sup>

firstname.lastname@iaik.tugraz.at

We revisit the AMBA AHB case study that has been used as a benchmark for several reactive synthesis tools. Synthesizing AMBA AHB implementations that can serve a large number of masters is still a difficult problem. We demonstrate how to use parameterized synthesis in token rings to obtain an implementation for a component that serves a single master, and can be arranged in a ring of arbitrarily many components. We describe new tricks – property decomposition synthesis, and direct encoding of simple GR(1) – that together with previously described optimizations allowed us to synthesize a component model with 14 states in about 1 hour.

## 1 Introduction

By automatically generating correct implementations from a temporal logic specification, reactive synthesis tools can relieve system designers from tedious and error-prone tasks like low-level manual implementation and debugging. This great benefit comes at the cost of high computational complexity of synthesis, which makes synthesis of large systems an ambitious goal. For instance, Bloem et al. [6] synthesize an arbiter for the ARM AMBA Advanced High Performance Bus (AHB) [2]. The results, obtained using RATSY [5], show that both the size of the implementation and the time for synthesis increase steeply with the number of masters that the arbiter can handle. This is unexpected, since an arbiter for  $n + 1$  masters is very similar to an arbiter for  $n$  masters, and manual implementations grow only slightly with the number of masters. While recent results show that synthesis time and implementation size can be improved in standard LTL synthesis tools [8, 10], the fundamental problem of increasing complexity with the number of masters can only be solved by adapting the synthesis approach itself.

To this end, Jacobs and Bloem [11] introduced the *parameterized synthesis* approach. In parameterized synthesis, we synthesize a component implementation that can be used as a building block, replicating components to form a system that satisfies a specification for *any* number of components. The approach is based on *cutoff* results that have previously only been used to reduce the *verification* of parameterized systems to systems of a small, fixed size. In particular, small cutoffs exist for token-ring networks, as shown by Emerson and Namjoshi [7]. These results can be extended to allow the reduction of the parameterized synthesis problem to a distributed synthesis problem with a fixed number of components, which can in turn be solved by a modification of the *bounded synthesis* procedure of Finkbeiner and Schewe [9]. As experiments with the original, naïve implementation of parameterized synthesis revealed that only very small specifications could be handled, Khalimov et al. [14] introduced a number of optimizations that improved runtimes for synthesis of token-ring systems by several orders of magnitude. In this paper, we will show how the resulting synthesis method can serve as the basis for synthesizing an implementation for the parameterized AMBA AHB specification.

\*The original version was submitted to SYNT 2014

<sup>†</sup>This work was supported by the Austrian Science Fund (FWF) under the RiSE National Research Network (S11406).

**Contributions.** We demonstrate how to synthesize a parameterized implementation of the AMBA AHB, with guaranteed correctness for any number of masters. To this end, we translate the LTL specification of the AMBA AHB (as found in [12]) into a version that is suitable for parameterized synthesis in token rings, and address several challenges with respect to theoretical applicability and practical feasibility:

1. We show how to *localize* global input and output signals, i.e., those that cannot be assigned to one particular master. This is necessary since our approach is based on the replication of components that act only on local information.
2. We introduce theoretical extensions of the cutoff results for token rings that support some of the features of the AMBA specification. This includes the handling (some but not all) of assumptions on local and global inputs, and of the fully asynchronous timing model (which includes the synchronous behavior intended in AMBA).
3. We show how to handle multiple process templates to support a *special process* with properties different from other processes.
4. We describe further *optimizations* that make synthesis feasible, in particular based on the insight that the AMBA protocol features three different types of accesses, and the control structures for these accesses can be synthesized (to some degree) independently.

Finally, we report on our practical experience with parameterized synthesis of the AMBA protocol, pointing out weaknesses and open problems in current synthesis approaches.

## 2 The AMBA Case Study

ARMs *Advanced Microcontroller Bus Architecture* (AMBA) [2] is a communication bus for a number of masters and clients on a microchip. The most important part of AMBA is the *Advanced High-performance Bus* (AHB), a system bus for the efficient connection of processors, memory, and devices.

The bus arbiter is the critical part of the AHB, ensuring that only one master accesses the bus at any time. Masters send HBUSREQ to the arbiter if they want access, and receive HGRANT if they are allowed to access it. Masters can also ask for different kinds of *locked transfers* that cannot be interrupted.

The exact arbitration protocol for AMBA is not specified. Our goal is to synthesize a protocol that guarantees safety and liveness properties. According to the specification, any device that is connected to the bus will react to an input with a delay of one time step. I.e., we are considering Moore machines. In the following, we introduce briefly which signals are used to realize the arbiter of this bus for masters.

**Requests and grants.** The identifier of the master which is currently active is stored in the  $n + 1$ -bit signal HMASTER[ $n:0$ ], with  $n$  chosen such that the number of masters fit into  $n + 1$  bits. To request the bus, master  $i$  raises signal HBUSREQ[ $i$ ]. The arbiter decides who will be granted the bus next by raising signal HGRANT[ $i$ ]. When the client raises HREADY, the bus access starts at the next tick, and there is an update  $\text{HMASTER}[n:0] := i$ , where HGRANT[ $i$ ] is currently active.

**Locks and bursts.** A master can request a locked access by raising both HBUSREQ[ $i$ ] and HLOCK[ $i$ ]. In this case, the master additionally sets HBURST[1:0] to either SINGLE (single cycle access), BURST4 (four cycle burst) or INCR (unspecified length burst). For a BURST4 access, the bus is locked until the client has accepted 4 inputs from the master (each signaled by raising HREADY). In case of a INCR access, the bus is locked until HBUSREQ[ $i$ ] is lowered. The arbiter raises HMASTLOCK if the bus is currently locked.

**LTL specification.** The original natural-language specification [2] has been translated into a formal specification in the GR(1) fragment of LTL before [12, 6, 10]. Figure 1 shows the environment assump-

Assumptions :		
	$G \quad (\text{HMASTERLOCK} \wedge \text{HBURST} = \text{INCR}) \rightarrow \text{XF} \neg \text{HBUSREQ}[\text{HMASTER}]$	(A1)
	$GF \quad \text{HREADY}$	(A2)
$\forall i :$	$G \quad \text{HLOCK}[i] \rightarrow \text{HBUSREQ}[i]$	(A3)
$\forall i :$	$\neg \text{HBUSREQ}[i] \wedge \neg \text{HLOCK}[i] \wedge \neg \text{HREADY}$	(A4)
Guarantees :		
	$G \quad \neg \text{HREADY} \rightarrow \text{X} \neg \text{START}$	(G1)
	$G \quad (\text{HMASTERLOCK} \wedge \text{HBURST} = \text{INCR} \wedge \text{START})$ $\rightarrow \text{X}(\neg \text{START} \text{W} (\neg \text{START} \wedge \text{HBUSREQ}[\text{HMASTER}]))$	(G2)
	$G \quad (\text{HMASTERLOCK} \wedge \text{HBURST} = \text{BURST4} \wedge \text{START} \wedge \text{HREADY})$ $\rightarrow \text{X}(\neg \text{START} \text{W} [3](\neg \text{START} \wedge \text{HREADY}))$	(G3.1)
	$G \quad (\text{HMASTERLOCK} \wedge \text{HBURST} = \text{BURST4} \wedge \text{START} \wedge \neg \text{HREADY})$ $\rightarrow \text{X}(\neg \text{START} \text{W} [4](\neg \text{START} \wedge \text{HREADY}))$	(G3.2)
$\forall i :$	$G \quad \text{HREADY} \rightarrow (\text{HGRANT}[i] \leftrightarrow \text{X}(\text{HMASTER} = i))$	(G4)
	$G \quad \text{HREADY} \rightarrow (\text{LOCKED} \leftrightarrow \text{X}(\text{HMASTERLOCK}))$	(G5)
$\forall i :$	$G \quad \text{X} \neg \text{START} \rightarrow \left( \begin{array}{l} (\text{HMASTER} = i \leftrightarrow \text{X}(\text{HMASTER} = i)) \\ \wedge \quad (\text{HMASTERLOCK} \leftrightarrow \text{X} \text{HMASTERLOCK}) \end{array} \right)$	(G6)
$\forall i :$	$G \quad (\text{DECIDE} \wedge \text{X} \text{HGRANT}[i]) \rightarrow (\text{HLOCK}[i] \leftrightarrow \text{X}(\text{LOCKED}))$	(G7)
$\forall i :$	$G \quad \neg \text{DECIDE} \rightarrow \left( \begin{array}{l} \text{HGRANT}[i] \leftrightarrow \text{X} \text{HGRANT}[i] \\ \wedge \quad \text{LOCKED} \leftrightarrow \text{X} \text{LOCKED} \end{array} \right)$	(G8)
$\forall i :$	$G \quad \text{HBUSREQ}[i] \rightarrow \text{F}(\neg \text{HBUSREQ}[i] \vee \text{HMASTER} = i)$	(G9)
$\forall i \neq 0 :$	$G \quad \neg \text{HGRANT}[i] \rightarrow (\neg \text{HGRANT}[i] \text{W} \text{HBUSREQ}[i])$	(G10.1)
	$G \quad (\text{DECIDE} \wedge (\forall i : \neg \text{HBUSREQ}[i])) \rightarrow \text{X} \text{HGRANT}[0]$	(G10.2)
	$\text{HGRANT}[0] \wedge (\forall i \neq 0 : \neg \text{HGRANT}[i]) \wedge \text{HMASTER} = 0 \wedge \neg \text{HMASTERLOCK}$ $\wedge \text{DECIDE} \wedge \text{START}$	(G11)

Figure 1: Formal specification of the AMBA AHB [12], in the GR(1) fragment of LTL.

tions and system guarantees from [12] that will be the basis for our parameterized specification. The full specification is  $(A1 \wedge \dots \wedge A4) \rightarrow (G1 \wedge \dots \wedge G11)$ .

### 3 Definitions

A *labeled transition system (LTS)* over sets  $O$  of output variables and  $I$  of input variables is a tuple  $(Q, Q_0, \Sigma, \delta, \lambda)$  where  $Q$  is the set of *states*,  $Q_0 \subseteq Q$  is the set of *initial states*,  $\Sigma = 2^I$  is the set of *inputs* (also called *transition labels*),  $\delta \subseteq Q \times \Sigma \times Q$  is the *transition relation*, and  $\lambda : Q \rightarrow 2^O$  is the *output function* (also called *state-labeling function*). Variables from  $O \cup I$  will be used as atomic propositions in our specifications.

#### 3.1 System Model

In this section we define the token ring system – the LTS that consists of replicated copies of a process connected in a uni-directional ring. Transitions in a token ring system are either internal or synchronized

(in which one process sends the token to the next process along the ring). The token starts in a non-deterministically chosen process.

Fix a set  $O_{pr}$  of (local) *output variables* that contain a distinguished output variable  $snd$ , and a set  $I_{pr}$  of (local) *input variables* that contain a distinguished input variable  $rcv$ .

**Process Template  $P$ .** Let  $\Sigma_{pr} = 2^{I_{pr}}$ . A *process template*  $P$  is a LTS  $(Q, Q_0, \Sigma_{pr}, \delta, \lambda)$  over  $O_{pr}$ :

- i) The state set  $Q$  is finite and can be partitioned into two non-empty disjoint sets:  $Q = T \cup NT$ . States in  $T$  are said to *have the token*.
- ii) The initial state set is  $Q_0 = \{t_t, t_n\}$  for some  $t_t \in T, t_n \in NT$ .
- iii) The output function  $\lambda : Q \rightarrow 2^{O_{pr}}$  satisfies that for every  $q \in NT$ ,  $snd \notin \lambda(q)$ .
- iv) Let  $\Sigma_{pr}^{rcv} = \{i \in \Sigma_{pr} \mid rcv \in i\}$ ,  $\Sigma_{pr}^{-rcv} = \Sigma_{pr} \setminus \Sigma_{pr}^{rcv}$ ,  $T^{snd} = \{q \in Q \mid snd \in \lambda(q)\}$ ,  $T^{-snd} = T \setminus T^{snd}$ . Then:

$$\delta \subseteq T^{snd} \times \Sigma_{pr}^{-rcv} \times NT \cup NT \times \Sigma_{pr}^{rcv} \times T \cup NT \times \Sigma_{pr}^{-rcv} \times NT \cup T^{-snd} \times \Sigma_{pr}^{-rcv} \times T.$$

Also,  $\delta$  is non-terminating: for every  $q \in NT$  and every  $in \in \Sigma_{pr}$  there exists  $q \xrightarrow{in} q' \in \delta$ ; and for every  $q \in T$  and every  $in \in \Sigma_{pr}^{-rcv}$  there exists  $q \xrightarrow{in} q' \in \delta$ .

- †) Given a fairness condition  $A_{loc}$  over  $I_{pr} \cup O_{pr}$ ,<sup>1</sup> we require that from any state  $q$  with the token, under any input sequence satisfying  $A_{loc}$ , the process will reach a state  $q'$  where it sends the token. We call this requirement  $\dagger$ .

**Ring Topology  $R$ .** A *ring* is a directed graph  $R = (V, E)$ , where the set of vertices is  $V = \{1, \dots, k\}$  for some  $k \in \mathbb{N}$ , and the set of edges is  $E = \{(i, i \oplus 1) \mid i \in V\}$ . Vertices are called *process indices*.

**Token-Ring System  $P^R$ .** Fix a ring topology  $R = (V, E)$ . Let  $I_{sys} = (I_{loc} \times V) \cup I_{glob}$  be the *system input variables*, where local inputs  $I_{loc}$  and global inputs  $I_{glob}$  are such that  $I_{pr} = I_{loc} \cup I_{glob}$ . Define  $\Sigma_{sys} = 2^{I_{sys}}$ . For system input  $in \in \Sigma_{sys}$ , let  $in(v) = \{i \in in \mid i \in I_{loc} \times \{v\} \cup I_{glob}\}$  denote the input to process  $v$  (including global inputs). Let  $O_{sys} = O_{pr} \times V$  be the *system output variables*. For  $(p, i)$  in  $O_{sys}$  or in  $I_{sys} \setminus I_{glob}$  we write  $p_i$ .

Given a process template  $P = (Q, Q_0, \Sigma_{pr}, \delta, \lambda)$  over  $O_{pr}$  and  $I_{pr}$  and a token ring topology  $R = (V, E)$ , define the *token-ring system*  $P^R$  as the finite LTS  $(S, S_0, \Sigma_{sys}, \Delta, \Lambda)$  over  $O_{sys}$  and  $I_{sys}$ , where:

- The set  $S$  of *global states* is  $Q^V$ , i.e., all functions from  $V$  to  $Q$ . If  $s \in Q^V$  is a global state then  $s(i)$  denotes the local state of the process with index  $i$ .
- The set of *global initial states*  $S_0$  contains all  $s_0 \in Q_0^V$  in which exactly one of the processes has the token.
- The labeling  $\Lambda(s) \subseteq O_{sys}$  for  $s \in S$  is defined as follows:  $p_i \in \Lambda(s)$  if and only if  $p \in \lambda(s(i))$ , for  $p \in O_{pr}$  and  $i \in V$ .

Finally, define the *global transition relation*  $\Delta$ . In a *fully asynchronous token ring*, a subset of all processes can make a transition in each step of the system, i.e.,  $\Delta$  consists of the following set of transitions:

- An *internal transition* is an element  $(s, in, s')$  of  $S \times \Sigma_{sys} \times S$  for which there are process indices  $M \subseteq V$  such that
  - i) for all  $v \in M$ :  $snd \notin \lambda(s(v))$  and  $rcv \notin in(v)$ ,

<sup>1</sup>Discussion of fairness conditions is deferred until Sect. 3.2.

- ii) for all  $v \in M$ :  $s(v) \xrightarrow{\text{in}(v)} s'(v)$  is a transition of  $P$ ,
- iii) for all  $u \in V \setminus M$ :  $s(u) = s'(u)$ .
- A *token-passing transition* is an element  $(s, \text{in}, s')$  of  $S \times \Sigma_{\text{sys}} \times S$  for which there are process indices  $M \subseteq V$  and two indices  $v, w \in M$  such that  $(v, w) \in E$  and
  - i)  $\text{snd} \in \lambda(s(v))$ , and  $\forall u \in M \setminus \{v\} : \text{snd} \notin \lambda(s(u))$  — i.e., only process  $v$  sends the token,
  - ii)  $\text{rcv} \in \text{in}(w)$  and for all  $u \in M \setminus \{w\} : \text{rcv} \notin \text{in}(u)$  — i.e., only process  $w$  receives the token,
  - iii) for every  $u \in M$ :  $s(u) \xrightarrow{\text{in}(u)} s'(u)$  is a transition of  $P$ ,
  - iv) for every  $u \in V \setminus M$ :  $s'(u) = s(u)$ .

Special cases of the fully asynchronous token ring are the *synchronous token ring* and the *interleaving token ring*. In a synchronous token ring,  $M = V$  for internal and token-passing transitions. I.e., always all processes make a transition simultaneously. In an interleaving token ring,  $M = \{v\}$  for some  $v \in V$  for internal transitions, and  $M = \{v, w\}$  for  $(v, w) \in E$  for token-passing transitions. I.e., at each moment either *exactly one* process makes an internal transition, or one process sends a token to the next process.

**System Run.** Fix a ring topology  $R = (V, E)$ . A *run of a token ring system*  $P^R = (S, S_0, \Sigma_{\text{sys}}, \Delta, \Lambda)$  over  $O_{\text{sys}}$  and  $I_{\text{sys}}$  is a finite or infinite sequence  $x = (s_1, \text{in}_1, M_1)(s_2, \text{in}_2, M_2) \dots$ , where:

- $s_1 \in S_0$ ,  $s_k \in S$  and  $\text{in}_k \in \Sigma_{\text{sys}}$  for any  $k \leq |x|$ ,
- for all  $k < |x| : (s_k, \text{in}_k, s_{k+1}) \in \Delta$ ,
- for all  $k < |x|$ :  $M_k$  is the set of processes transiting in  $(s_k, \text{in}_k, s_{k+1})$  (see  $M$  in the definition of  $\Delta$ ).

### 3.2 Parameterized Systems and Specifications

The *parameterized ring* is the function  $\mathcal{R} : n \mapsto \mathcal{R}(n)$ , where  $n \in \mathbb{N}$  and  $\mathcal{R}(n)$  is the ring with  $n$  vertices. A *parameterized token ring system* is a function  $P^{\mathcal{R}} : n \mapsto P^{\mathcal{R}(n)}$ , where  $n \in \mathbb{N}$  and  $P$  is a given process template. When necessary to disambiguate we explicitly write ‘parameterized fully asynchronous token ring systems’ or ‘parameterized interleaving token ring systems’.

A *parameterized specification* is a sentence in indexed temporal logic, that is, a temporal logic formula with indexed variables and quantification over indices. Variables are from the set of output and input variables  $O_{\text{pr}} \cup I_{\text{pr}}$ , and indices refer to different copies of process templates. A parameterized token ring system  $P^{\mathcal{R}}$  *satisfies* a parameterized specification  $\phi$ , written  $P^{\mathcal{R}} \models \phi$ , iff  $\forall n: P^{\mathcal{R}(n)} \models \phi$ . This definition assumes a fixed semantics for temporal logic formulas in labeled transition systems. Below, we introduce a slightly non-standard semantics as a modification of the (action-based) semantics in Emerson and Namjoshi [7] to the case of open systems.

**Semantics for Open Systems.** Emerson and Namjoshi [7] consider closed systems (i.e., without inputs), but with transitions labeled by *actions* that can also be used in specifications. In the semantics defined below, we simulate an action  $a$  by an input corresponding to  $a$ . Furthermore, for defining when a given process satisfies a formula, we consider the projection of the run onto those points in time where the process actually makes a transition, just like actions are only considered when a transition fires.

In addition, we extend our semantics to the fully asynchronous timing model, which in particular includes the synchronous timing model that is needed for reasoning about the AMBA case study.<sup>2</sup> This

<sup>2</sup>Note that in the synchronous timing model, our semantics is the same as the standard semantics (every process always makes a transition, so all inputs are considered), but we need the fully asynchronous case for the cutoff result in Thm. 3. Intuitively, in synchronous systems an implementation can count the number of global steps until it receives the token again, and therefore correctness of such implementations may depend on the size of the ring, making cutoff results impossible. Thus, systems that are correct in the synchronous but not in the fully asynchronous case are of limited interest to us.

leads to additional problems: the natural ways to extend the semantics to properties of more than one process is to consider either the projection to those points in time where *at least one* of the processes makes a step, or to those where *all* processes make a step. Both cases are undesirable: in the first case, we also consider inputs that the processes cannot read, and in the second case the property will only be guaranteed at those points in time where all processes make a step together — which is clearly undesirable, e.g., for a mutual exclusion property. Therefore, in properties that talk about more than one process, we do not allow input signals at all.

Fix a token ring system  $P^R = (S, S_0, \Sigma_{\text{sys}}, \Delta, \Lambda)$  over  $O_{\text{sys}}$  and  $I_{\text{sys}}$ . We describe the semantics of parameterized properties 1-indexed properties over  $O_{\text{sys}}$  and  $I_{\text{sys}}$ , and 2-indexed properties over  $O_{\text{sys}}$ .

**Semantics of 1-indexed properties.** *1-indexed properties* are of the form  $\forall i. \varphi(i)$ , where  $\varphi(i)$  is an LTL formula over system variables that are indexed with  $i$ .

Fix a process index  $j$ . Given a run  $(s_1, \text{in}_1, M_1)(s_2, \text{in}_2, M_2) \dots$ , consider the sub-sequence that contains exactly those  $(s_k, \text{in}_k, M_k)$  with  $j \in M_k$ . The *local run of process  $j$*  is obtained by mapping each  $(s_k, \text{in}_k, M_k)$  in the sub-sequence to  $(s_k(j), \text{in}_k(j))$ , where  $s_k(j)$  is the local state of  $j$  and  $\text{in}_k(j)$  are inputs to  $j$ . Then, *a system run satisfies  $\varphi(j)$*  iff the local run of process  $j$  satisfies  $\varphi(j)$ . The latter satisfaction is defined in the usual way. Note that since we consider only elements of the system run where process  $j$  makes a step, we can use the next-time operator  $X$  (interpreted locally, cp. [14, Sect. 5.2][7, Sect. 2.5]).

**Example.** Consider a typical 1-indexed property of an arbiter  $\forall i. G(r_i \rightarrow F g_i)$  (‘for every process, every request should be finally granted’). In the semantics of 1-indexed properties described above, this property should be read as: ‘for every process, every request *that has been seen by the process* should be finally granted’. Another example: in the new semantics the property  $\forall i. G(r_i \rightarrow X g_i)$  should be read as ‘for every process, every request that has been seen by the process should be granted the next step’. Notice that the environment cannot falsify the property by not scheduling the process.

**Semantics of 2-indexed properties.** *2-indexed properties* are of the form  $\forall i, j. \varphi(i, j)$ , where  $\varphi(i, j)$  is an  $\text{LTL} \setminus X$  formula over output variables (and no input variables) that are indexed with  $i$  or  $j$ . Satisfaction for fixed process indices  $i, j$  is defined in the standard way: A system run  $(s_1, \text{in}_1, M_1)(s_2, \text{in}_2, M_2) \dots$  *satisfies*  $\varphi(i, j)$  iff the sequence  $\Lambda(s_1)\Lambda(s_2) \dots$  satisfies  $\varphi(i, j)$ . I.e., we consider all elements of the system run, no matter if processes  $i$  and  $j$  make the transition or not.

**Example.** Consider a typical 2-indexed property of the mutual exclusion  $\forall i \neq j. G \neg(g_i \wedge g_j)$ . In the semantics of 2-indexed properties described above, the property should be read in a usual way: ‘it is never the case that two processes grant at the same time.’

**Semantics of  $A_{\forall i. \text{ass}(i)} \varphi$ .** Let  $\varphi$  be a 1- or 2-indexed property as introduced above. *A system satisfies  $A_{\forall i. \text{ass}(i)} \varphi$*  iff any system run that satisfies  $\forall i. \text{ass}(i)$  also satisfies  $\varphi$ , where satisfaction of 1-indexed  $\forall i. \text{ass}(i)$  and of  $\varphi$  as defined above.

**Note on the Semantics of GR(1).** The AMBA specification [12] is defined in the GR(1) fragment of LTL, where the implication between assumptions and guarantees is usually interpreted with a special semantics [15]. In this paper, we instead use the standard semantics for this implication.

### 3.3 Parameterized Synthesis Problem

The *parameterized synthesis problem* in token rings is: given a parameterized specification  $\varphi$ , find an implementation  $P$  such that  $P^{\mathcal{R}} \models \varphi$ . The problem is in general undecidable:

**Theorem 1** ([11], Theorem 3.5). *The parameterized synthesis problem of interleaving token rings with no global inputs is undecidable for specifications  $\forall i, j. A \varphi(i, j)$ , where  $A \varphi(i, j)$  is an  $\text{LTL} \setminus X$  formula over processes  $i, j$ .*

The proof reduces the undecidable problem of distributed synchronous synthesis [17] to distributed synthesis of an interleaving token ring of size 2, which implies undecidability of parameterized synthesis.

Note that Theorem 1 does not apply to specifications of the form  $A_{\forall i. \text{ass}(i)} \forall j. \varphi(j)$ . In fact, we can use the hub-abstraction technique (see [14, Section 6]) to prove the following:

**Observation 1.** *The parameterized synthesis problem of token rings without global inputs is decidable for specifications  $\forall j. A_{\forall i. \text{ass}(i)} \varphi(j)$  where  $\text{ass}(i)$  and  $\varphi(i)$  are LTL formulas over process  $i$ .*

## 4 The Existing Parameterized Synthesis Approach

The parameterized synthesis problem in token rings is in general undecidable, but Jacobs and Bloem [11] have introduced a semi-decision procedure for the problem. It is based on i) the cutoff results of [7], which state that model checking parameterized token rings is equivalent to model checking token rings of a cutoff size, and ii) the *bounded synthesis* method [9] that turns an undecidable synthesis problem (of synthesizing a token ring of fixed size) into a possibly infinite sequence of decidable synthesis problems (of synthesizing a token ring of fixed size in which a process implementation is not larger than the bound) by iterative bounding the size of process implementations.

### 4.1 Cutoff Results in Token Rings

A *cutoff* for a parameterized specification  $\varphi$  is a number  $c \in \mathbb{N}$  s.t.  $P^{\mathcal{R}(c)} \models \varphi \iff \forall n \geq c : P^{\mathcal{R}(n)} \models \varphi$ .

**Theorem 2** ([7], Theorem 3). *For interleaving parameterized token ring systems with no global inputs and parameterized specifications  $\forall i. A_{\forall i. \text{ass}(i)} \varphi(i)$ , where  $\varphi(i)$  and  $\text{ass}(i)$  are LTL formulas over process  $i$ , the cutoff is 2.*

**Corollary 1** ([11]). *The parameterized synthesis problem of interleaving token rings with no global inputs for parameterized specifications  $\forall i. A_{\forall i. \text{ass}(i)} \varphi(i)$ , where  $\varphi(i)$  and  $\text{ass}(i)$  are LTL formulas over process  $i$ , can be reduced to the synthesis problem of the token ring of size 2.*

### 4.2 Bounded Synthesis Method

By bounding the desired size of implementations, *bounded synthesis* [9] reduces the synthesis problem to a sequence of SMT problems. Uninterpreted functions are used to describe the transition relation, output functions, and auxiliary ‘ranking’ functions, and the SMT solver tries to find valuations of these functions such that the specification is satisfied. The flow is the following:

1. **Automata translation:** The negation of a given specification  $\varphi$  is translated into a non-deterministic Büchi automaton  $A_{\neg\varphi}$ .
2. **SMT encoding:** We encode a ranking function  $\rho$  on states of the product of the specification automaton  $A_{\neg\varphi}$  and the uninterpreted system. Consider a transition from composed state  $(q, s)$  to  $(q', s')$ , where  $q, q'$  are states of  $A_{\neg\varphi}$  and  $s, s'$  are states of the system. Then we require  $\rho(q', s') > \rho(q, s)$  if  $q'$  is an accepting state of  $A_{\neg\varphi}$ , and otherwise  $\rho(q', s') \geq \rho(q, s)$ . The rule ensures that the SMT constraints are satisfiable if and only if the product does not have loops with an accepting state of the automaton, and a solution represents a correct implementation of the system.
3. **SMT solving, iteration for increasing bounds:** If the SMT constraints in step 2 are satisfiable, then return the implementation. Otherwise, there exists no implementation of the given size bound; we increase the bound and repeat step 2.

For the details of the SMT encoding that we use for synthesis of token rings see Khalimov et al. [14].

## 5 Challenges for the Existing Approach

Based on the existing approach for parameterized synthesis, we want to synthesize a system that satisfies

$$A((A1 \wedge \dots \wedge A4 \wedge \text{FairSched}) \rightarrow (G1 \wedge \dots \wedge G11 \wedge TR)),$$

where

- *FairSched* is the form  $\forall i. GFsch_i$ , specifying that every process is scheduled infinitely often.
- *TR* are guarantees ensuring that the process template satisfies the requirements of the token ring process template defined in Sect. 3.1:

$$\begin{aligned} \forall i. \quad & G(\text{SEND}[i] \rightarrow \text{TOK}[i]) \\ \forall i. \quad & G(\text{TOK}[i] \wedge \neg \text{SEND}[i] \rightarrow X\text{TOK}[i]) \\ \forall i. \quad & G(\neg \text{TOK}[i] \wedge \neg \text{SEND}[i-1] \rightarrow X\neg \text{TOK}[i]) \\ \forall i. \quad & G(\text{TOK}[i] \rightarrow F\text{SEND}[i]) \end{aligned}$$

As the existing cutoff results of Emerson and Namjoshi [7] (or their extensions by Aminof et al. [1]) do not support all features of the AMBA specification, we need to address the following challenges:

1. **Synchronous AMBA and global inputs:** The AMBA protocol uses synchronous timing and has several global inputs (that are shared between all processes), while the cutoff results in [7, 1] are for interleaving systems with local action labels instead of inputs. We have discussed in Sect. 3.2 how actions simulate local inputs, but global inputs are not supported in the existing cutoff theorems. In Sect. 6.1 we extend the cutoff results to fully asynchronous token rings with global inputs. For our synthesis approach, we will use the fact that correctness of an implementation in the fully asynchronous case implies correctness in the synchronous case.
2. **Global outputs:** The AMBA specification assumes that there are global outputs, i.e., those that depend on the global state of the system, such as HMASTLOCK. This is not handled by [7, 1], and in Sect. 6.2 we address this by synthesizing local outputs that can be manually converted to suitable global outputs with simple logical operations.
3. **Special 0-process, immediate reaction, global information:** The AMBA specification distinguishes between master number 0 and all other masters. We support this by synthesizing two different process implementations, one that serves master 0, and one for all other processes. Furthermore, process 0 is supposed to immediately grant master 0 when no process receives a HBUS-REQ[i] signal - this is a problem since only processes that have the token should give a grant, and information about requests of other processes is not available to process 0. We show how to handle this by weakening the specification and introducing an auxiliary global input in Sect. 6.3.

## 6 Obtaining and Handling a Parameterized AMBA Specification

In the following, we will show how we obtained a parameterized AMBA specification suitable to our parameterized synthesis approach, and how we extended the approach to handle this specification.

### 6.1 Addressing Challenges ‘Synchronous AMBA’ and ‘Global Inputs’

We will first extend the cutoff results for token rings to fully asynchronous systems with global inputs, for restricted classes of process templates and assumptions. Since these classes are not sufficient to model AMBA, we will afterwards introduce a method to localize assumptions in a sound but incomplete way.



### 6.1.1 Complete Approach: New Cutoff Results

We consider systems and specifications that satisfy the following assumptions:

- a)  $P = (Q, Q_0, \Sigma_{pr}, \delta, \lambda)$  is such that:  $\forall q \in Q$  with  $\text{snd} \in \lambda(q)$  there exists unique  $q' \in Q$  such that  $q \xrightarrow{\text{in}} q'$  for any input  $\text{in} \in \Sigma_{pr}$ . I.e., in all sending states the process ignores inputs.
- b) The assumptions  $\forall i. \text{ass}(i)$  are of the form  $\forall i. G \alpha(i)$  or of the form  $\forall i. \alpha(i)$ , where  $\alpha(i)$  is a Boolean formula over inputs (including global inputs) of process  $i$ .

Then (proofs can be found in the appendix):

**Theorem 3.** *Assume conditions (a) and (b). Then, for parameterized fully asynchronous token ring systems and parameterized specifications as stated below, the cutoffs are:*

- for  $\forall i. A_{\forall i G \alpha(i)} \phi(i)$  the cutoff is 2,
- for  $\forall i, j. A_{\forall i G \alpha(i)} \psi(i, j)$  the cutoff is 4,

where  $\alpha(i)$  is a Boolean formula over inputs of process  $i$ ,  $\phi(i)$  is an LTL formula over inputs and outputs of process  $i$ , and  $\psi(i, j)$  is an  $LTL \setminus X$  formula over outputs of processes  $i, j$ .

Note that the problem becomes undecidable if we do not restrict fair path properties, i.e., if we remove (b) but still assume (a):

**Observation 2.** *The parameterized model checking problem for fully asynchronous token rings with global inputs and properties of the form  $\forall i. A_{\forall i. \text{ass}(i)} \phi(i)$  is undecidable, where  $\text{ass}(i)$  and  $\phi(i)$  are LTL formulas over inputs and outputs of process  $i$  (including global inputs).*

Note that Theorem 3 does not support all assumptions in the AMBA specification (Fig. 1): A3 and A4 are supported by the theorem, but A1 and A2 are not.

### 6.1.2 Incomplete Approach: Localization of Assumptions

Since Theorem 3 does not support assumptions A1 and A2, we introduce an approach that *localizes* the assumptions, essentially rewriting the specification  $\forall j. A_{\forall i. \text{ass}(i)} \phi(j)$  into a form  $\forall j. A(\text{ass}(j) \rightarrow \phi(j))$ .<sup>3</sup>

However, this naive form of localization strengthens the AMBA specification too much, making it unrealizable. Instead, we use a specialized way for localizing assumptions in token rings.

**Localization of assumptions in token rings.** As suggested in [14, Sect.6],

$$A_{\forall i. \text{ass}(i)} \forall j. (\text{gua}(j) \wedge TR(j))$$

is localized into

$$\forall i. A(\text{ass}(i) \rightarrow TR(i)) \wedge (\text{ass}(i) \wedge GF \text{TOK}[i] \rightarrow \text{gua}(i)),$$

where  $\text{ass}(i)$  includes *FairSched*, and  $TR$  are the token ring properties as defined in Sect. 5.

This restores realizability in our case. Intuitively, this specification guarantees that  $TR$  will be satisfied under the given local assumptions, and for the rest of the guarantees we can then assume that all other processes will eventually send the token, thus satisfying the additional assumption  $GF \text{TOK}[i]$ .

**Linking token possession to mutual exclusion.** In addition to global assumptions, the original specification contains an implicit mutual exclusion property: G4 defines how HMASTER is updated by the HGRANT[i] signals. Note that G4 can only be satisfied if the HGRANT[i] are mutually exclusive. Since

<sup>3</sup>Note that in some cases, the localized version is equivalent to the original one, e.g. for A2, since HREADY is a global input.

we know that the token can (and must) be used to ensure mutual exclusion, we explicitly specify this by adding G12:  $\forall i. \text{HGRANT}[i] \rightarrow \text{TOK}[i]$ . Together with localization of assumptions, this ensures that the parameterized specification will be 1-indexed.

**Resulting specification.** The resulting specification is of the form

$$\forall i. A((\text{ass}(i) \rightarrow \text{TR}(i)) \wedge ((\text{ass}(i) \wedge \text{GF TOK}[i]) \rightarrow \text{gua}(i) \wedge \text{G12})),$$

i.e., a 1-indexed LTL property in prenex-indexed form.

While Theorem 3 supports some formulas of the type  $A_{\forall i. \text{ass}(i)} \forall j. \text{gua}(j)$ , solving the synthesis problem for formulas with assumptions in this form is costly. In particular, every liveness assumption introduces a loop (with length equal to the size of the ring under consideration) for every liveness guarantee in the specification. This severely blows up the size of the specification automaton. Thus, even for the liveness assumptions A3 and A4 that are supported by the theorem, we use the *localization* approach.

## 6.2 Addressing Challenge ‘Global Outputs’

To address this challenge we define what is a localizable global output, introduce a special version of localizable global outputs we use for AMBA, and modify the specification to handle these global outputs.

**Linking global to local outputs.** For a given parameterized system, a *localizable global output* is a global output that can be expressed as a propositional formula over terms of the form  $\forall i. \alpha(i)$  and  $\exists i. \alpha(i)$ , where  $\alpha(i)$  is a propositional formula over outputs of process  $i$ .

**Fixed solution for AMBA.** The AMBA specification in Fig. 1 has global outputs HMASTLOCK, START, DECIDE, and HMASTER. We restrict synthesis to search for a solution with a fixed localizable implementation of global outputs, namely: For each global output signal  $g$  we introduce a local output signal  $g_i$ , and define

- $\text{HMASTER} := i$  whenever  $\text{HMASTER}[i]$  is high, and
- $g := \exists i. \text{TOK}[i] \wedge g_i$  for all other global outputs  $g$ .

**Modification of the parameterized specification.** According to the two previous steps, we should replace all global outputs in the specification with their specialized localizable definitions in terms of the new local outputs. For example, START should be replaced by  $\exists i. \text{TOK}[i] \wedge \text{START}[i]$ . However, in token ring systems the only communication between processes is token passing, and hence the value of  $\exists i. \text{TOK}[i] \wedge \text{START}[i]$  is not known to a process, except when it has the token (and thus defines that value). Thus, we replace each global output with its local version, e.g., START is replaced by  $\text{START}[i]$ .

Note that the limited communication interface (via token passing) does not make AMBA unrealizable, even though processes cannot access the value of global outputs when they do not possess. Intuitively, this is because the token is the shared resource that guarantees mutual exclusion of grants, and therefore the values of these global signals should always be controlled by the process that has the token. In particular, outputs DECIDE and START are signals that are used to decide when to raise a grant and when to start and end a bus access<sup>4</sup>, which should only be done when the token is present. Similarly, signals HMASTLOCK and HMASTER should be controlled by the process that currently controls the bus (and hence has the token). By using only the local version of these signals in the specification, we force the implementation to never raise them unless the process has the token.

<sup>4</sup>The original AMBA specification [2] does not have these signals – they were introduced to simplify the formalization of the specification [12].

### 6.3 Addressing Challenges ‘Special 0-process’ and ‘Global Information’

The AMBA specification is of the form  $A_{\forall i.ass(i)}(\forall i \neq 0. \varphi(i) \wedge \psi(0))$ , i.e., it distinguishes the behavior of process 0. Recall the AMBA guarantees G10 from Fig. 1 (after localization steps of the previous sections):

$$\forall i \neq 0: G \neg HGRANT[i] \rightarrow (\neg HGRANT[i] \mathcal{W} HBUSREQ[i]) \quad (G10.1)$$

$$G (\text{DECIDE}[0] \wedge (\forall i: \neg HBUSREQ[i])) \rightarrow X HGRANT[0] \quad (G10.2)$$

The distinction between 0- and non-0-processes, as well as the required properties, present several additional challenges to the parameterized synthesis approach.

**Distinguished 0-process.** The process templates for 0- and non-0-processes for specifications of the form  $A_{\forall i.ass(i)}(\forall i \neq 0. \varphi(i) \wedge \psi(0))$  can be synthesized separately, i.e., first, synthesize a process template  $P_\varphi$  for  $A_{\forall i.ass(i)} \forall i. \varphi(i)$  and a process template  $P_\psi$  for  $A_{\forall i.ass(i)} \forall i. \psi(i)$ . Then a combined token ring consisting of any number of copies of  $P_\varphi$  and of one copy of  $P_\psi$  at 0 vertex will satisfy  $A_{\forall i.ass(i)}(\forall i \neq 0. \varphi(i) \wedge \psi(0))$ . Hence we introduce a separate specification for the 0-process and synthesize it separately.

To this end, we also separate G11 into two parts, G11.1:  $\neg HGRANT[i] \wedge \neg HMASTERLOCK[i]$  (for non-0-processes) and G11.2:  $\text{TOK}[0] \rightarrow HGRANT[0] \wedge HMASTER[0] \wedge \neg HMASTERLOCK[0]$  (for 0-process).

**Immediate reaction.** Guarantee G10.2 requires an immediate reaction to a state where no process receives a bus request. This is unrealizable for AMBA in token rings because mutual exclusion of the grants requires possession of the token and implies  $G(HGRANT[i] \rightarrow \text{TOK}[i])$ . To allow the process to wait for the token and then immediately react, we modify G10.2 to  $G (\neg \text{TOK}[0] \wedge X \text{TOK}[0] \wedge \forall i: \neg HBUSREQ[i]) \rightarrow X HGRANT[0]$ <sup>5</sup>.

**Global information.** G10.2 contains an index quantifier  $\forall i$  inside the temporal operator  $G$ , which is not supported by Thm. 3. Intuitively, G10.2 requires 0-process to have *global information* about inputs of all processes, as it needs to react to a situation where  $\text{HBUSREQ}[i]$  is low for all  $i$ . This is not possible when only  $\text{HBUSREQ}[0]$  is available as an input, so we introduce an auxiliary (global) input  $\text{NO\_REQ}$ , and add the assumption  $\forall i. G(\text{HBUSREQ}[i] \rightarrow \neg \text{NO\_REQ})$ . Then G10.2 becomes:  $G (\neg \text{TOK}[0] \wedge X \text{TOK}[0] \wedge \text{NO\_REQ}) \rightarrow X HGRANT[0]$ . Such guarantees and assumptions are allowed by Thm. 3.

### 6.4 Resulting Parameterized AMBA Specification

We obtained the new specification from the one in Fig. 1 by localization of global assumptions (Sect. 6.1), localization of global output signals  $\text{HMASTER}$ ,  $\text{HMASTERLOCK}$ ,  $\text{DECIDE}$ , and  $\text{START}$  (Sect. 6.2), and separation of specifications for 0- and non-0-processes (Sect. 6.3). The resulting assumptions and guarantees for non-0-processes are given in Fig. 2, the modifications for the 0-process in Fig. 3. The specifications to be synthesized are

$$\forall i. A((A1 \wedge \dots \wedge A4 \rightarrow \text{TR}(i)) \wedge (A1 \wedge \dots \wedge A5 \rightarrow G1 \wedge \dots \wedge G10.1 \wedge G11.1 \wedge G12)), \text{ and}$$

$$\forall i. A((A1 \wedge \dots \wedge A4 \wedge A6 \rightarrow \text{TR}(i)) \wedge (A1 \wedge \dots \wedge A6 \rightarrow G1 \wedge \dots \wedge G10.2 \wedge G11.2 \wedge G12)).$$

<sup>5</sup>You probably expect to see  $G (\text{DECIDE}[0] \wedge (\forall i: \neg \text{HBUSREQ}[i]) \wedge X \text{TOK}[0]) \rightarrow X HGRANT[0]$ , but this makes the specification unrealizable due to the token ring requirement  $G(\text{TOK} \rightarrow \text{FSEND})$ : the environment falsifies it by making  $\forall i: \neg \text{HBUSREQ}[i]$  true whenever the process raises  $\text{DECIDE}$  (and hence the process should continue granting and cannot release the token). To regain realizability one could add additional assumptions (something like  $G F(\text{DECIDE} \wedge \neg \text{NO\_REQ})$ ). Instead, we decided to change slightly the semantics of the specification.

Local Assumptions :		
G	$((\text{HMASTLOCK}[i] \wedge (\text{HBURST} = \text{INCR}) \wedge \text{HMASTER}[i]) \rightarrow \text{XF} \neg \text{HBUSREQ}[i])$	(A1)
G	$\text{FHREADY}$	(A2)
G	$\text{HLOCK}[i] \rightarrow \text{HBUSREQ}[i]$	(A3)
	$\neg \text{HBUSREQ}[i] \wedge \neg \text{HLOCK}[i] \wedge \neg \text{HREADY}$	(A4)
G	$\text{FTOK}[i]$	(A5)
Local Guarantees :		
G	$\neg \text{HREADY} \rightarrow \text{X} \neg \text{START}[i]$	(G1)
G	$(\text{HMASTLOCK}[i] \wedge \text{HBURST} = \text{INCR} \wedge \text{START}[i]) \rightarrow \text{X}(\neg \text{START}[i] \text{W} (\neg \text{START}[i] \wedge \text{HBUSREQ}[i]))$	(G2)
G	$(\text{HMASTLOCK}[i] \wedge \text{HBURST} = \text{BURST4} \wedge \text{START}[i] \wedge \text{HREADY}) \rightarrow \text{X}(\neg \text{START}[i] \text{W} [3](\neg \text{START}[i] \wedge \text{HREADY}))$	(G3.1)
G	$(\text{HMASTLOCK}[i] \wedge \text{HBURST} = \text{BURST4} \wedge \text{START}[i] \wedge \neg \text{HREADY}) \rightarrow \text{X}(\neg \text{START}[i] \text{W} [4](\neg \text{START}[i] \wedge \text{HREADY}))$	(G3.2)
G	$\text{HREADY} \rightarrow (\text{HGRANT}[i] \leftrightarrow \text{XHMASTER}[i])$	(G4)
G	$\text{HREADY} \rightarrow (\text{LOCKED}[i] \leftrightarrow \text{XHMASTLOCK}[i])$	(G5)
G	$\text{X} \neg \text{START}[i] \rightarrow \left( \begin{array}{l} \text{HMASTER}[i] \leftrightarrow \text{XHMASTER}[i] \\ \wedge \text{HMASTLOCK}[i] \leftrightarrow \text{XHMASTLOCK}[i] \end{array} \right)$	(G6)
G	$(\text{DECIDE}[i] \wedge \text{XHGRANT}[i]) \rightarrow (\text{HLOCK}[i] \leftrightarrow \text{XLOCKED}[i])$	(G7)
G	$\neg \text{DECIDE}[i] \rightarrow \left( \begin{array}{l} \text{HGRANT}[i] \leftrightarrow \text{XHGRANT}[i] \\ \wedge \text{LOCKED}[i] \leftrightarrow \text{XLOCKED}[i] \end{array} \right)$	(G8)
G	$\text{HBUSREQ}[i] \rightarrow \text{F}(\neg \text{HBUSREQ}[i] \vee \text{HMASTER}[i])$	(G9)
G	$\neg \text{HGRANT}[i] \rightarrow (\neg \text{HGRANT}[i] \text{W} \text{HBUSREQ}[i])$	(G10.1)
	$\neg \text{HGRANT}[i] \wedge \neg \text{HMASTLOCK}[i]$	(G11.1)
G	$\text{HGRANT}[i] \rightarrow \text{TOK}[i]$	(G12)

Figure 2: Parameterized AMBA specification for non-0-processes. G10.2 is only needed for 0-process.

Local Assumptions :	<i>as before:</i>	A1,A2,A3,A4,A5	
	<i>new:</i>	G HBUSREQ[i] $\rightarrow \neg \text{NO\_REQ}$	(A6)
Local Guarantees :	<i>as before:</i>	G1,G2,G3,G4,G5,G6,G7,G8,G9,G12	
	<i>removed:</i>	G10.1,G11.1	
	<i>new:</i>	G(NO_REQ $\wedge \neg \text{TOK}[i] \wedge \text{XTOK}[i]) \rightarrow \text{XHGRANT}[i]$	(G10.2)
	<i>modified:</i>	TOK[i] $\rightarrow \text{HGRANT}[i] \wedge \text{HMASTER}[i] \wedge \neg \text{HMASTLOCK}[i]$	(G11.2)

Figure 3: Parameterized AMBA specification for 0-process: modifications wrt. non-0-processes.

## 7 Optimizations and Experiments

In this section, we describe optimizations that proved to be crucial for the synthesis of the parameterized AMBA AHB, and present the results of parameterized synthesis in form of runtimes and resulting component implementations.

**Prototype.** The basis of our experiments is PARTY, a tool for parameterized synthesis of token rings [13]. PARTY is written in Python, uses LTL3BA [3] for automata translation and Z3 [16] for SMT solving. All experiments were done on a x86\_64 machine with 2.60GHz, 12GB RAM. Prototype implementation and specification files can be found at <https://github.com/5nizza/Party/> (branch ‘amba-gr1’).

**Synchronous Hub Abstraction [14, Sect.6].** Synchronous hub abstraction can be applied to 1-indexed specifications. It lets the environment simulate all but one process, and always schedules this process. Thus, the synthesizer searches for a process template in synchronous setting with additional assumptions on the environment, namely: i) the environment sends the token to the process infinitely often, and ii) the environment never sends the token to the process if it already has it. Note that synchronous hub abstraction is sound and complete for the semantics of 1-indexed properties introduced in Sect. 3.2. Also note that after applying this optimization any monolithic synthesis method can be applied to the resulting specification in Sect. 6.4.

**Hardcoding States With and Without the Token [14, Sect.4].** The number of states with and without the token in a process template defines the degree of the parallelism in a token ring. Parallelism increases with the number of states that do not have the token. In the AMBA case study, any action with grant depends on having the token. Thus we divide states into one that does not have the token, and all others that have the token, by hardcoding the TOK[i] output function.

**Decompositional Synthesis of Different Grant Schemes.** The idea of the decompositional synthesis is: synthesize a subset of the properties, then synthesize a larger subset using the model from the previous step as basis. Consider an example of the synthesis of the non-0-process of AMBA. The flow is:

1. Assume that every request is locked with BURST4, i.e., add the assumption  $G(HLOCK[i] \wedge HBURST = BURST4)$  to the specification. This implicitly removes guarantee G2 and assumption A1 from the specification. Synthesize the model. The resulting model has 10 states (states  $t0, \dots, t9$  and transitions between them in Fig. 4).
2. Use the model found in the previous step as a basis: assert the number of states, values of output functions in these states, transitions for inputs that satisfy the previous assumption. Transitions for inputs that violate the assumption from step 1 are not asserted, and thus left to be synthesized.

Now relax the assumptions: allow locked and non-locked BURST4 requests, i.e., replace the previous assumption with  $G(HBURST = BURST4)$ . Again, this implicitly removes G2 and A1. In contrast to the last step, now guarantee G3 is not necessarily ‘activated’ if there is a request.

Synthesize the model. This may require increasing the number of states (and it does in the case of non-0 process) – add new states and keep assertions on all the previous states.

3. Assert the transitions of the model found, like in the previous step.

Remove all added assumptions and consider the original specification. Synthesize the final model.

Although for AMBA this approach was successful, it is not clear how general it is. For example, it does not work if we start with locked BURST4 and HREADY always high, and then try to relax it. Also, the separation into sets of properties to be synthesized was done manually.

**Optimization of SMT Encoding.** Recall from Sect. 4 that SMT based bounded synthesis, given an automaton  $A_{\neg\varphi}$  of the negation of specification  $\varphi$  and an unknown process template  $P = (Q_P, Q_0, \Sigma_{pr}, \delta_P, out)$

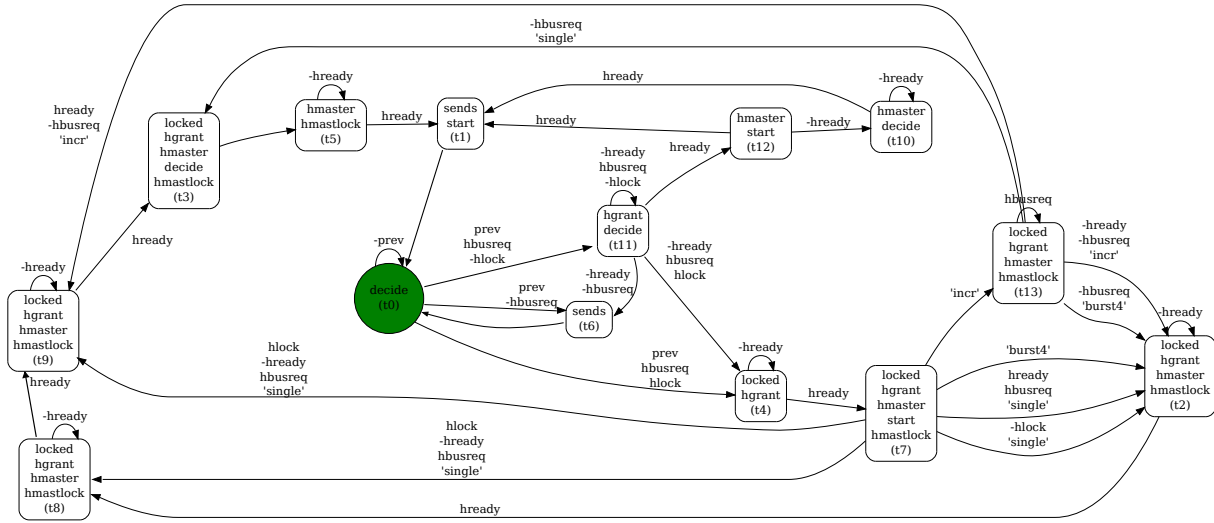


Figure 4: Synthesized model of non-0-processes (after manual simplification). Circle green state ( $t_0$ ) is without the token, other states are with the token. Initial states are  $t_0, t_1$ . States are labeled with their active outputs. Edges are labeled with inputs, a missing input variable means “don’t care”. ‘Burst4’ means  $HBURST = BURST4$ , ‘incr’ means  $HBURST = INCR$ , ‘single’ means neither of them. In the first step of compositional synthesis states  $t_0, \dots, t_9$  were synthesized, in the second  $t_{10}, \dots, t_{12}$  were added, in the final step state  $t_{13}$  was added.

with a fixed number of states, encodes the product automaton  $A_{-\varphi} \times P$  into SMT constraints such that  $A_{-\varphi} \times P$  contains no reachable loops with an accepting state of the  $A_{-\varphi}$  iff SMT constraints are satisfiable. Below is a general assertion from which the SMT query is composed:

$$\bigwedge q \in Q_P \bigwedge a \xrightarrow{i,o} b \in \delta_{A_{-\varphi}} : \rho(a, q) \geq 0 \wedge o = out(q) \rightarrow \rho(b, \delta_P(q, i)) \triangleright \rho(a, q)$$

where  $\triangleright$  is ‘>’ if  $b$  is an accepting state of  $A_{-\varphi}$ , else ‘ $\geq$ ’. In words: for any state of the process template, and any transition of the automaton, if the current state of the product automaton is reachable, then the next state should also be reachable and the ranking function should be as stated.

The specification of AMBA we synthesize is derived from GR(1) specification. As a consequence it contains assumptions (A3, A6) of the form  $G\alpha(i)$  where  $\alpha(i)$  is a Boolean formula over current inputs, and many guarantees (G1, G4, G5, G6, G7, G8, G12, G10.2) of the form  $G\beta(i, o, o')$  where  $\beta(i, o, o')$  is a Boolean formula over current inputs and outputs and next outputs. Instead of using the standard approach via automaton translation described above, we:

1. encode assertions of the form  $G\alpha(i)$  directly into SMT constraints, namely add  $\alpha(i)$  to the the premise of the SMT rule. Thus, the premise becomes ‘ $\rho(a, q) \geq 0 \wedge o = out(q) \wedge \alpha(i) \rightarrow \dots$ ’
2. for all guarantees of the form  $G\beta(i, o, o')$  add SMT constraints of the form:

$$\bigwedge q \in Q_P \bigwedge i \in \Sigma_{pr} : \alpha(i) \rightarrow \beta(i, out(q), out(\delta_P(q, i)))$$

The first optimization is sound and complete, the second one introduces incompleteness.

For AMBA specification in Fig. 2 and 3 this optimization means that only guarantees G2, G3, G9, G10.1, G11 require the standard flow via automata translation.

Does this optimization help in the synthesis? Preliminary experiments (considering the first step of the compositional synthesis of non-0 process) show:

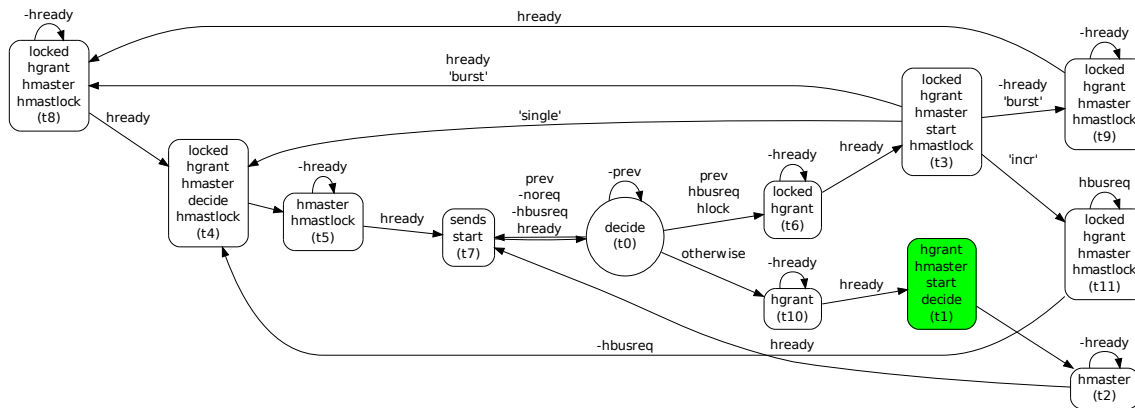


Figure 5: Synthesized model of 0-processes (after manual simplification). Circle state ( $t0$ ) is without the token, other states are with the token. Initial states are  $t0, t1$ . States are labeled with their active outputs. Edges are labeled with inputs, a missing input variable means “don’t care”. ‘Burst’ means  $\text{HBURST} = \text{BURST3}$  (recall we decreased the length of bursts for 0 process), ‘incr’ means  $\text{HBURST} = \text{INCR}$ , ‘single’ means neither of them. In the first step of compositional synthesis states  $t0, \dots, t10$  were synthesized, in the second only transitions were synthesized, but no new states added, in the final step  $t11$  was added.

- With the optimization the automaton for the negated specification has 24 states, without – 42 states.
- The synthesis time with optimization is 16 minutes, without – 57 minutes. Interesting to note that the optimized and non-optimized versions spent the same time (2 minutes) checking satisfiability of the last query (with the model size of 10), so the main difference is in checking unsatisfiable queries – Z3 identifies unsatisfiability of optimized queries faster (14 vs. 53 minutes). A similar behavior happens for a version of the same specification with reduced lengths of bursts ( $3/4 \rightarrow 2/3$ ): total times are 3/6 minutes, but the last query took 1m40s/30s for optimized/non-optimized version.

**Results.** Synthesis times are in Tables 1 and 2, the model synthesized for non-0-process is in Fig. 4. The table has timings for the case when all optimizations described in this section are enabled — it was not our goal to evaluate the optimizations separately, but to find a combination that works for the AMBA case study.

For the 0-process we considered a simpler version with burst lengths reduced to 2/3 instead of the original 3/4 ticks. With the original length the synthesizer could not find a model within 2 hours (it hanged checking 11 states models while the model has at least 12 states).

Without the decompositional approach, the synthesizer could not find a model for for non-0 process of the AMBA specification within (at least) 5 hours.

## 8 Conclusions

We have shown that parameterized synthesis in token rings can be used to solve benchmark problems of significant size, in particular the well-known AMBA AHB specification that has been used as a synthesis

Table 1: Results for non-0-process.

Additional assumptions	time	#states
GHLOCK	16min.	10
GHBURST = BURST4	13sec.	13
– (Full Specification)	1min.	14

Table 2: Results for 0-process  
(bursts reduced:  $3/4 \rightarrow 2/3$ ).

Additional assumptions	time	#states
GHLOCK	3h.	11
GHBURST = BURST4	1min.	11
– (Full Specification)	1m30s.	12

benchmark for a long time. To achieve this goal, we extended slightly the cutoff results that parameterized synthesis is based on, and used a number of optimizations in the translation of the specification and the synthesis procedure itself to make the process feasible.

This is the first time that the AMBA case study, or any other realistic case study of significant size, has been solved by an automatic synthesis procedure for the general, parametric case. However, some of the steps in the procedure are manual or use an ad-hoc solution for the specific problem at hand, like the limited extension of cutoff results for global inputs, the construction of suitable functions to convert local to global outputs, or the decomposition synthesis for different grant schemes. Generalizing and automating these approaches is left for future work.

Furthermore, our synthesized implementation is such that the size of the parallel composition grows only linearly with the number of components. Thus, for this case study our approach does not only solve the problem of increasing synthesis time for a growing number of components, but also the problem of implementations that need an exponential amount of memory in the number of components. We pay for this small amount of memory with a less-than-optimal reaction time, as processes have to wait for the token in order to grant a request. This restriction could be remedied by extending the parameterized synthesis approach to different system models, e.g., processes that coordinate by guarded transitions, or communicate via broadcast messages.

**Acknowledgments.** We thank Sasha Rubin for insightful discussions on cutoff results in token rings.

## References

- [1] Benjamin Aminof, Swen Jacobs, Ayrat Khalimov & Sasha Rubin (2014): *Parameterized Model Checking of Token-Passing Systems*. In: *VMCAI, LNCS 8318*, Springer, pp. 262–281, doi:10.1007/978-3-642-54013-4\_15.
- [2] ARM Ltd. (1999): *AMBA Specification (Rev.2)*. Available from [www.arm.com](http://www.arm.com).
- [3] Tomáš Babiak, Mojmír Kretínský, Vojtech Reháč & Jan Strejcek (2012): *LTL to Büchi Automata Translation: Fast and More Deterministic*. In: *TACAS, LNCS 7214*, Springer, pp. 95–109, doi:10.1007/978-3-642-28756-5\_8.
- [4] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.
- [5] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan & Richard Seeber (2010): *RATSY - A New Requirements Analysis Tool with Synthesis*. In: *CAV, LNCS 6174*, Springer, pp. 425–429, doi:10.1007/978-3-642-14295-6\_37.
- [6] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Yaniv Sa’ar (2012): *Synthesis of Reactive(1) designs*. *J. Comput. Syst. Sci.* 78(3), pp. 911–938, doi:10.1016/j.jcss.2011.08.007.
- [7] E. Allen Emerson & Kedar S. Namjoshi (2003): *On Reasoning About Rings*. *Int. J. Found. Comput. Sci.* 14(4), pp. 527–550, doi:10.1142/S0129054103001881.



- [8] Bernd Finkbeiner & Swen Jacobs (2012): *Lazy Synthesis*. In: *VMCAI, LNCS 7148*, Springer, pp. 219–234, doi:10.1007/978-3-642-27940-9\_15.
- [9] Bernd Finkbeiner & Sven Schewe (2013): *Bounded synthesis*. *STTT* 15(5-6), pp. 519–539, doi:10.1007/s10009-012-0228-z.
- [10] Yashdeep Godhal, Krishnendu Chatterjee & Thomas A. Henzinger (2013): *Synthesis of AMBA AHB from formal specification: a case study*. *STTT* 15(5-6), pp. 585–601, doi:10.1007/s10009-011-0207-9.
- [11] Swen Jacobs & Roderick Bloem (2014): *Parameterized Synthesis*. *Logical Methods in Computer Science* 10, pp. 1–29, doi:10.2168/LMCS-10(1:12)2014.
- [12] Barbara Jobstmann (2007): *Applications and Optimizations for LTL Synthesis*. Ph.D. thesis, Graz University of Technology.
- [13] Ayrat Khalimov, Swen Jacobs & Roderick Bloem (2013): *PARTY Parameterized Synthesis of Token Rings*. In Natasha Sharygina & Helmut Veith, editors: *CAV, LNCS 8044*, Springer, pp. 928–933, doi:10.1007/978-3-642-39799-8\_66. Available at <http://dx.doi.org/10.1007/978-3-642-39799-8>.
- [14] Ayrat Khalimov, Swen Jacobs & Roderick Bloem (2013): *Towards Efficient Parameterized Synthesis*. In: *VMCAI, LNCS 7737*, Springer, pp. 108–127, doi:10.1007/978-3-642-35873-9\_9.
- [15] Uri Klein & Amir Pnueli (2010): *Revisiting Synthesis of GR(1) Specifications*. In: *Haifa Verification Conference, LNCS 6504*, Springer, pp. 161–181, doi:10.1007/978-3-642-19583-9\_16.
- [16] Leonardo de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *TACAS, LNCS 4963*, Springer, pp. 337–340, doi:10.1007/978-3-540-78800-3\_24.
- [17] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *FOCS*, IEEE Computer Society, pp. 746–757, doi:10.1109/FSCS.1990.89597.

## 9 Appendix

### 9.1 Proof of Theorem 3

*Proof idea.* Consider the second case, the first one is similar. For an arbitrary process template  $P$  that satisfies (a), we need to prove that:

$$\forall n \geq c. P^{\mathcal{R}(n)} \models \forall i, j. A_{\forall i. \text{ass}(i)} \psi(i, j) \iff P^{\mathcal{R}(c)} \models \forall i, j. A_{\forall i. \text{ass}(i)} \psi(i, j).$$

The  $\implies$  direction is trivial since the left part includes  $P^{\mathcal{R}(c)}$ . The  $\impliedby$  direction can be rewritten as

$$\exists n \geq c. P^{\mathcal{R}(n)} \models \exists i, j. E_{\forall i. \text{ass}(i)} \neg \psi(i, j) \implies P^{\mathcal{R}(c)} \models \exists i, j. E_{\forall i. \text{ass}(i)} \neg \psi(i, j).$$

The proof uses the notion of stuttering bisimulation [4] and consists of two ideas: symmetry argument and simulation construction.

**Symmetry argument:** To prove the cutoff results for  $\exists i, j. E_{\forall i. \text{ass}(i)} \neg \psi(i, j)$  it is enough to prove it for  $\exists j. E_{\forall i. \text{ass}(i)} \neg \psi(0, j)$ .

Suppose that  $\exists i, j. E_{\forall i. \text{ass}(i)} \neg \psi(i, j)$  is satisfied in a token ring of size  $n$ . Consider  $i, j \in \{0, \dots, n-1\}$  that satisfy  $E_{\forall i. \text{ass}(i)} \neg \psi(i, j)$ , and assume for simplicity  $i < j$ . Consider a rotated token ring: process 0 in the rotated ring repeats the behavior of process  $i$  in the original token ring, process 1 repeats the behavior of process  $i \oplus 1$ , and so on. Then, the rotated ring will satisfy the property  $E_{\forall i. \text{ass}(i)} \neg \psi(0, j-i)$ .

**Simulation construction:** if there is a run of a large system that satisfies  $\exists j. E_{\forall i. \text{ass}(i)} \neg \psi(0, j)$  then there is a run of the cutoff system that satisfies it.

For any  $k \in \{2, \dots, n-2\}$ :

$$\exists j. E_{\forall i. \text{ass}(i)} \neg \psi(0, j) \iff E_{\forall i. \text{ass}(i)} \neg \psi(0, 1) \vee E_{\forall i. \text{ass}(i)} \neg \psi(0, k) \vee E_{\forall i. \text{ass}(i)} \neg \psi(0, n-1).$$

Intuitively, the statement above holds because in a fully asynchronous token ring, a 2-indexed property can only ‘identify’ whether the processes are direct neighbors or not, but not the number of processes in between. Thus, if  $E_{\forall i. G \alpha(i)} \neg \psi(0, k)$  holds for some  $k \in \{2, \dots, n-2\}$ , then it holds for any  $k \in \{2, \dots, n-2\}$ . Let  $k = 2$ , and thus consider

$$\exists j. E_{\forall i. G \alpha(i)} \neg \psi(0, j) \iff E_{\forall i. G \alpha(i)} \neg \psi(0, 1) \vee E_{\forall i. G \alpha(i)} \neg \psi(0, 2) \vee E_{\forall i. G \alpha(i)} \neg \psi(0, n-1).$$

We will analyze the case where  $E_{\forall i. G \alpha(i)} \neg \psi(0, 2)$  is satisfied in the large ring. Other cases are similar.

We show how processes 0 and 2 in the cutoff ring (of size 4) simulate transitions of corresponding processes of the ring of size 5, see Fig. 6. We assume for simplicity that process 0 starts with the token. A similar construction works in the general case for a ring of size  $n$ . The construction, given a run of the large ring that satisfies  $\forall i. G \alpha(i)$  and  $\neg \psi(0, 2)$ , builds a run of the cutoff ring that satisfies them:

- Until ‘moment 1’ the system run of the large ring is repeated exactly in the cutoff ring.
- At ‘moment 1’, process 3 of the large token ring receives the token – in the cutoff ring there is no corresponding process. Hence, we stutter process 2 in the cutoff ring in the state with the token, while other processes move as they move in the large token ring from moment 1 to 2.
- Between moments 1 and 2, in the large ring process 2 may move without the token. To simulate these transitions, in the cutoff ring process 2 sends the token to process 3 and then repeats the transitions of process 2 of the large ring between moments 1 and 2, while all other processes stutter. We need to repeat these transitions to ensure the states of all processes of the cutoff ring equal to states of the corresponding processes in the large ring (in the beginning of moment 1)<sup>6</sup>. Thus, the path between moments 1 and 2 may be longer in the cutoff ring.
- The problematic step in the figure is at moment 2. In the large ring, process 2 sends the token to process 3 and reads a ‘red’ global input, while in the cutoff ring the construction provides a (possibly) different ‘black’ input to the other processes. This is where we use the additional restrictions (a) and (b) on process template and formulas for fair paths – they allow replacement of ‘red’ with ‘black’ input, such that (a) the behavior of the process does not change, and (b) the property will still be satisfied.
- Finally, from moment 2 until process 0 receives the token, the cutoff ring repeats transitions of the large token ring exactly. Then, the new round starts and we repeat the construction.

*Correctness.* The construction ensures: i) local runs<sup>7</sup> of all the processes of the cutoff ring satisfy assumptions  $\forall i. G \alpha(i)$  if they were satisfied in the large ring, ii) global run, projected on outputs of processes 0 and 2 in the cutoff ring, is equal up to stuttering to that of processes 0 and 2 in the large ring<sup>8</sup>. These two items imply that the system run of the cutoff ring satisfies  $\forall i. ass(i)$  and  $\neg \psi(0, 2)$  if the system run of the large ring satisfies  $\forall i. ass(i)$  and  $\neg \psi(0, 2)$ .  $\square$

## 9.2 Proof of Observation 2

*Proof idea.* In [7, Section 6] the authors prove that if the token is allowed to carry a single bit of information, then the parameterized model checking (even when a process template has no inputs) problem becomes undecidable. Global inputs and fair path assumptions can be used to simulate token values (which implies the undecidability of the parameterized model checking) in the following way:

<sup>6</sup>Strictly speaking, the exact same global state (if we ignore the abstracted process) is not required for proving the theorem.

<sup>7</sup>Recall that local run of process  $i$  considers only the elements of the system run in which process  $i$  moves, see Sect. 3.

<sup>8</sup>The construction does not ensure that global runs projected on outputs *and* inputs are the same if consider inputs in all moments of the system run, i.e., even when processes  $i, j$  do not move.

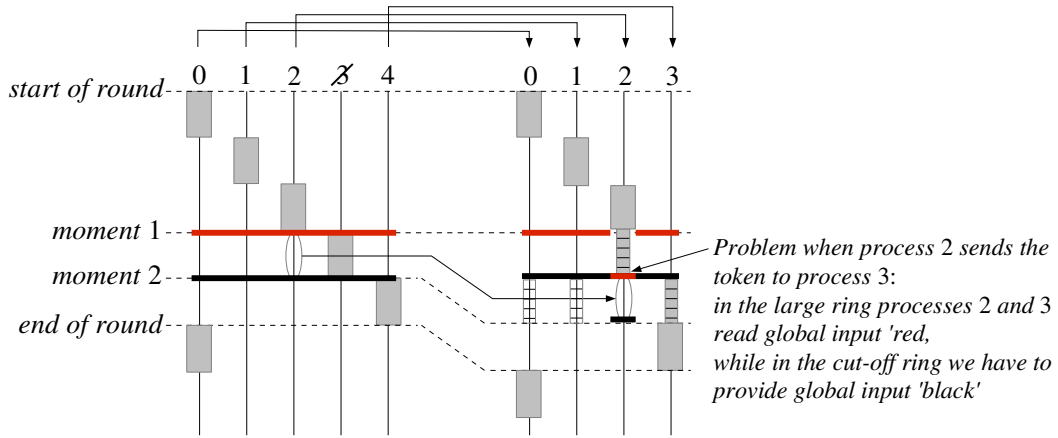


Figure 6: Simulating a system run of the large token ring of size 5 by a cutoff token ring of size 4. Behavior of processes 0,1,2,4 in the large ring is simulated by processes 0,1,2,3 respectively in the cutoff ring (behavior of process 3 of the large token ring is abstracted away). The run starts at the top, and evolves downwards. Vertical solid lines denote process states (exact states are omitted). Gray boxes denote states with the token, lined boxes mean stuttering (processes do not move). Bold red and black horizontal lines denote different global inputs the processes received at this step.

- The process template has two states  $snd_0, snd_1$  that send the token:  $snd \in \lambda(snd_0) \wedge snd \in \lambda(snd_1)$
- $ass(i) := G(snd_0 \rightarrow glob_0) \wedge G(snd_1 \rightarrow glob_1) \wedge G \neg(glob_0 \wedge glob_1)$ , i.e., only paths on which the environment provides global input  $glob_i$  when process sends the token from state  $snd_i$  are considered.

This process template ensures: on paths that satisfy  $\forall i. G ass(i)$ , if a process receives the token and reads global input  $glob_i$ , then the previous process sent the token from state  $snd_i$ . This way a (value-less) token ring with global inputs and special assumptions on fair paths can model rings with the binary token.  $\square$